Dynamic list scheduling of threads on clusters

G. G. H. Cavalheiro, E. D. Benitez, D. S. Perancon, E. Moschetta

UNISINOS
Universidade do Vale do Rio dos Sinos
Programa Interdisciplinar de Pós Graduação em Computação Aplicada
Overview

• Introduction
• Anahy
  – Task and synchronizations
  – Programming interface
  – Scheduling strategy
• Handling a Graph of Tasks
  – Visualizing an execution
• Some Performances
• The Future of Anahy
Introduction

- Performance portability

-- The concurrency of an application can be described regardless of hardware resources
Introduction

- **Performance portability**
- **Concurrency**
  - Depends on application characteristics
  - Can be identified by a specialist on the application
- **Parallelism**
  - Depends on hardware
  - A specialist on applications is not necessarily an specialist in parallel programming

Concurrency >> Parallelism
Introduction

• Performance portability

• Our approach:
  – Dissociate programming of execution

• Our proposal:
  – ANAH

• Our mechanisms:
  – Scheduling and dataflow control achieved at run time
Anahy

- Environment

  API

  Performance
  portability

  Applicative scheduling

  HW/OS dependent
  modules

  Execution pool
  multithreading

  Communication
  Active Messages

  Operating System
  Hardware

  Generic architecture

  Programming interface

DSM 2006
Anahy

Task and Synchronization

- A *task* defines a sequence of instructions and two set of data: input and output data;
- The synchronization between tasks are guaranteed by accesses to the data.
Task and Synchronization

- A task defines a sequence of instructions and two set of data: input and output data;
- The synchronization between tasks are guaranteed by accesses to the data.

Large amount of concurrency

large amount of synchronizations
Task and Synchronization

- A *task* defines a sequence of instructions and two sets of data: input and output data;
- The *synchronization* between tasks are guaranteed by accesses to the data.

Large amount of concurrency

large amount of synchronizations

Coarse scheduling unity: a thread
Anahy

- **Execution pool**
  - A set of *system* threads is responsible for executing the *athreads*
  - Each system thread is called VP
  - **Strategy:**
    - A VP can choose a specific *athread* to execute

List of ready *athreads*

getAnyReadyWork()
Anahy

- **Execution pool**
  - A set of *system* threads is responsible for executing the athreads
  - Each system thread is called VP
  - **Strategy:**
    - A VP can chose a specific athread to execute
    - The list of ready works is organized as a graph of dependencies

- **Graph of ready athreads**
  - `getTheWork(id)`
  - `getAnyReadyWork()`
  - `getTheWork(id)`
Anahy

- Programming Interface
- Creation
  
  ```c
  int athread_create( athread_t *th,
                     athread_attr_t *attrib,
                     void *(*func) (void *),
                     void *in );
  ```

- Synchronization
  
  ```c
  int athread_join( athread_t th, void **res );
  ```

- Athread code
  
  ```c
  void *foo( void *in ) {
    ...
    return out;
  }
  ```
Anahy

- Programming Interface

```c
void* foo(void* x) {
    ...
}
void* bar(void* p) {
    Task_A
    t1 = create(foo,a);
    Task_B
    t2 = create(fuu,b);
    ...
    join(t1,r1)
    Task_C
    join(t2,r2)
    Task_D
    return &something;
}
```

A thread (executing bar)

A thread (Executing foo)

Task

(create(foo,a))

join(t1,r1)

Task (code executed between two synchronizations)
Anahy

- Scheduling
- List scheduling
  - Blind strategy
    - Explosion on concurrency or memory
- Scheduling heuristics
  - Different searches on the graph
- Applied:
  - When a VP becomes idle and request for work
  - When is executed a join operation
Handling graph of tasks

• Search an athread on the graph:
  – athread_t* SearchFrom(from, direction, orientation, axis)

a.Join(b)
Handling graph of tasks

- Search an athread on the graph:
  - athread_t* SearchFrom(from, direction, orientation, axis)

  a.Join( b )

  Starts a new independent flow

  Helps the execution
Handling graph of tasks

• Examples
  – SearchFrom( current, ROOT, LEFT, VERT )
    • returns the next athread ready in the sub-graph having current as root (left-to-right, high priority on deep nodes)
  – SearchFrom( NULL, TOP, RIGHT, HORIZ )
    • returns the next athread ready in the graph from the first node of the graph (right-to-left, high priority on high nodes).
  – SearchFrom( jid, ROOT, RIGHT, HORIZ )
    • returns the next athread ready in the sub-graph having jid as root (right-to-left, high priority on the higest athread in the sub-graph).
Handling graph of tasks

- **Visual example**
  - **Recursive program:**
    ```c
    void* tree( void* n ) {
    if( n > 2 ) {
      t1 = create( tree, *n-1 );
      t2 = create( tree, *n-2 );
      doSomething( ... );
      join(t1,&r1);
      join(t2,&r2);
    }
    else doSomething( ... );
    return &something;
    }
    ```

- **VP idle:**
  - searches the last created in the highest level
    SearchFrom( NULL, TOP, RIGHT, HORIZ )

- **athread blocked in a join:**
  - searches a ready athread from jid
    SearchFrom( jid, HERE, RIGHT, HORIZ )
Performance

High Parallel Application
Ratio: Cilk / Anahy on a dual-processor

<table>
<thead>
<tr>
<th>Depth</th>
<th>Concurrency level on the program</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 VP</td>
<td></td>
</tr>
<tr>
<td>2 VPs</td>
<td></td>
</tr>
<tr>
<td>3 VPs</td>
<td></td>
</tr>
<tr>
<td>4 VPs</td>
<td></td>
</tr>
<tr>
<td>5 VPs</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VPs</th>
<th>Parallel execution support</th>
</tr>
</thead>
<tbody>
<tr>
<td>d = 15</td>
<td></td>
</tr>
<tr>
<td>d = 16</td>
<td></td>
</tr>
<tr>
<td>d = 19</td>
<td></td>
</tr>
<tr>
<td>d = 22</td>
<td></td>
</tr>
<tr>
<td>d = 25</td>
<td></td>
</tr>
<tr>
<td>d = 28</td>
<td></td>
</tr>
<tr>
<td>d = 29</td>
<td></td>
</tr>
</tbody>
</table>
Performance

High Parallel Application

Execution times: Athapascan-1 x Anahy on a cluster

Parallel execution support per node
The future of Anahy

• Current work
  – Distributed version
  – Real applications
    • Dynamic programming
    • Metabolic cellular network
    • Crowd simulation

• Next
  – Scheduling strategies

• Next++
  – Other Pthreads synchronization mechanisms
    • Mutex, condition variables
Dynamic list scheduling of threads on clusters

G. G. H. Cavalheiro, E. D. Benitez, D. S. Peranconhi, E. Moschetta

gersonc@anahy.org, anahy@anahy.org